University of Pittsburgh

Anonymous Functions

Week 3

MEMS 1140—Introduction to Programming in Mechanical Engineering

Learning Objectives (L.O.)

At the end of this lecture, you should understand/be able to:

- ❑ What functions are;
- ❑ What anonymous functions are;
- ❑ Write anonymous functions;
- ❑ Determine when to use an anonymous function;
- ❑ Apply an anonymous function to the dot product.

Table of Contents (ToC)

1 – A Strange Introduction to Functions

To introduce a practical definition for functions in programming, consider the following question:

What steps are involved in determining the color of a rose?

For us, this could be as simple as visual observation.

But given a large pile of roses to sort, it could be beneficial to prepare a machine to do it automatically.

1 – The Identification Steps

An identification algorithm could involve the following steps:

1. Illuminating a rose under a white light;
2. Measuring the reflected wavelengths;
3. Converting the reflection spectrum into an RGB color code.

This algorithm describes a human-agnostic approach to identifying a rose's color.

1 – Relation to Programming

With a well-defined algorithm, a machine can be prepared to automatically reapply the aforementioned steps in order to sort a large pile of roses.

With such a machine, the operator no longer has to consider every individual step; they can just turn on the machine.

This would be a mechanical analog of a ***function*** in code, perhaps named **`sortRoseByColor(rose)`**.

1 – So...What are Functions?

A function encapsulates a set of instructions that accomplish a specific task.

The purpose of writing functions is to make a program more modular, especially when operations are repeated many times.

With them, it is easier to read, debug, and maintain code.

2 – What are Anonymous Functions?

Anonymous functions are one-line operations that can be called from anywhere within the script in which it is defined.

An anonymous function is used to extract a specific, repeated piece of the script into a distinct unit ***within*** a script.

That operation can then be executed by using the function, rather than rewriting the same code over and over again.

3 – How to Write Anonymous Functions

The syntax for declaring an anonymous function is as follows:

```
<variable_name>
```

3 – How to Write Anonymous Functions

The syntax for declaring an anonymous function is as follows:

```
<variable_name> = @(<input_1>)
```

3 – How to Write Anonymous Functions

The syntax for declaring an anonymous function is as follows:

```
<variable_name> = @(<input_1>, <input_2>, etc.)
```

3 – How to Write Anonymous Functions

The syntax for declaring an anonymous function is as follows:

```
<variable_name> = @(<input_1>, <input_2>, etc.) <function_operation>;
```

For example, an anonymous function that returns the square root of the input to the sixth power would be written as follows:

```
fancy_function = @(x)
```

3 – How to Write Anonymous Functions

The syntax for declaring an anonymous function is as follows:

```
<variable_name> = @(<input_1>, <input_2>, etc.) <function_operation>;
```

For example, an anonymous function that returns the square root of the input to the sixth power would be written as follows:

```
fancy_function = @(x) sqrt(x^6);
```

This stores the operation ***in the variable*** `fancy_function`.

3 – Calling an Anonymous Function

Once it has been declared, the function can be accessed by calling `variable_name`, just like any other variable.

```
fancy_function = @(x) sqrt(x^6); % define the function
.
.
```

MATLAB Script (.m)

3 – Calling an Anonymous Function

Once it has been declared, the function can be accessed by calling `variable_name`, just like any other variable.

```
fancy_function = @(x) sqrt(x^6);
a = 2;
.
```

MATLAB Script (.m)

3 – Calling an Anonymous Function

Once it has been declared, the function can be accessed by calling `variable_name`, just like any other variable.

```
fancy_function = @(x) sqrt(x^6);
a = 2;
b = fancy_function(a)                  % apply the function
```

MATLAB Script (.m)

```
b = 8                                % 2^6 = 64 -> sqrt(64) = 8
```

Command Window Output

4 – Determining a Good Application

In general, functions are recommended when an operation needs to be reused multiple times.

Anonymous functions, in particular, are good for defining *short* operations.

As we have seen, it is very simple to define one line of math that needs to be calculated repeatedly.

5 – Application to the Dot Product

The vector dot product is perfect for an anonymous function!

The math is *really* simple; $\vec{v}_1 \cdot \vec{v}_2$ is one line of algebra:

$$\vec{v}_1 \cdot \vec{v}_2 = (v_{1,x} v_{2,x}) + (v_{1,y} v_{2,y}) + (v_{1,z} v_{2,z})$$

This can be easily implemented with a function that accepts two **`1x3`** vectors (like those discussed in Lecture 1.4) as inputs.

```
dot_product = @(v1, v2)
```

5 – Application to the Dot Product

The vector dot product is perfect for an anonymous function!

The math is *really* simple; $\vec{v}_1 \cdot \vec{v}_2$ is one line of algebra:

$$\vec{v}_1 \cdot \vec{v}_2 = (v_{1,x} v_{2,x}) + (v_{1,y} v_{2,y}) + (v_{1,z} v_{2,z})$$

This can be easily implemented with a function that accepts two `1x3` vectors (like those discussed in Lecture 1.4) as inputs.

```
dot_product = @(v1, v2) (v1(1)*v2(1)) +
```

5 – Application to the Dot Product

The vector dot product is perfect for an anonymous function!

The math is *really* simple; $\vec{v}_1 \cdot \vec{v}_2$ is one line of algebra:

$$\vec{v}_1 \cdot \vec{v}_2 = (v_{1,x} v_{2,x}) + (v_{1,y} v_{2,y}) + (v_{1,z} v_{2,z})$$

This can be easily implemented with a function that accepts two **`1x3`** vectors (like those discussed in Lecture 1.4) as inputs.

```
dot_product = @(v1, v2) (v1(1)*v2(1)) + (v1(2)*v2(2)) +
```

5 – Application to the Dot Product

The vector dot product is perfect for an anonymous function!

The math is *really* simple; $\vec{v}_1 \cdot \vec{v}_2$ is one line of algebra:

$$\vec{v}_1 \cdot \vec{v}_2 = \left(v_{1,x} v_{2,x}\right) + \left(v_{1,y} v_{2,y}\right) + \left(v_{1,z} v_{2,z}\right)$$

This can be easily implemented with a function that accepts two **`1x3`** vectors (like those discussed in Lecture 1.4) as inputs.

```
dot_product = @(v1, v2) (v1(1)*v2(1)) + (v1(2)*v2(2)) + (v1(3)*v2(3));
```

5 – Using the Dot Product

With the function, it is straightforward to use it to solve for the dot product between any two vectors:

```
dot_product = @(v1, v2) (v1(1)*v2(1)) + (v1(2)*v2(2)) + (v1(3)*v2(3));
.
.                    .
.                    .
.
.
.
```

MATLAB Script (.m)

5 – Using the Dot Product

With the function, it is straightforward to use it to solve for the dot product between any two vectors:

```
dot_product = @(v1, v2) (v1(1)*v2(1)) + (v1(2)*v2(2)) + (v1(3)*v2(3));

vector_1 = [1, 2, 3]; .
.                     .
.
.
.
```

MATLAB Script (.m)

5 – Using the Dot Product

✓ L.O.1
✓ L.O.2
✓ L.O.3
✓ L.O.4
⇨ L.O.5

With the function, it is straightforward to use it to solve for the dot product between any two vectors:

```
dot_product = @(v1, v2) (v1(1)*v2(1)) + (v1(2)*v2(2)) + (v1(3)*v2(3));

vector_1 = [1, 2, 3]; vector_2 = [4, 5, 6];
.                     .
.
.
.
```

MATLAB Script (.m)

5 – Using the Dot Product

With the function, it is straightforward to use it to solve for the dot product between any two vectors:

```
dot_product = @(v1, v2) (v1(1)*v2(1)) + (v1(2)*v2(2)) + (v1(3)*v2(3));

vector_1 = [1, 2, 3]; vector_2 = [4, 5, 6];
vector_3 = [7, 8, 9]; .
.
.
.
```

MATLAB Script (.m)

5 – Using the Dot Product

With the function, it is straightforward to use it to solve for the dot product between any two vectors:

```
dot_product = @(v1, v2) (v1(1)*v2(1)) + (v1(2)*v2(2)) + (v1(3)*v2(3));

vector_1 = [1, 2, 3]; vector_2 = [4, 5, 6];
vector_3 = [7, 8, 9]; vector_4 = [10, 11, 12];
.
.
.
```

MATLAB Script (.m)

5 – Using the Dot Product

With the function, it is straightforward to use it to solve for the dot product between any two vectors:

```
dot_product = @(v1, v2) (v1(1)*v2(1)) + (v1(2)*v2(2)) + (v1(3)*v2(3));

vector_1 = [1, 2, 3]; vector_2 = [4, 5, 6];
vector_3 = [7, 8, 9]; vector_4 = [10, 11, 12];

dot_product_result_12 = dot_product(vector_1, vector_2)  % v1 w/ v2
.
```

MATLAB Script (.m)

5 – Using the Dot Product

With the function, it is straightforward to use it to solve for the dot product between any two vectors:

```
dot_product = @(v1, v2) (v1(1)*v2(1)) + (v1(2)*v2(2)) + (v1(3)*v2(3));

vector_1 = [1, 2, 3]; vector_2 = [4, 5, 6];
vector_3 = [7, 8, 9]; vector_4 = [10, 11, 12];

dot_product_result_12 = dot_product(vector_1, vector_2)
dot_product_result_34 = dot_product(vector_3, vector_4)  % v3 w/ v4
```

MATLAB Script (.m)

5 – Dot Product Results

The results of both dot products are computed as follows:

```
dot_product_result_12 =
    32
.
.
```

Command Window Output

5 – Dot Product Results

The results of both dot products are computed as follows:

```
dot_product_result_12 =
    32
dot_product_result_34 =
    266
```

Command Window Output

Rather than writing the same code twice, it is good practice to write it once in a function and to use that instead.

6 – Summary

✓ L.O.1
✓ L.O.2
✓ L.O.3
✓ L.O.4
✓ L.O.5

This lecture covered:

✓ What functions are

Functions are powerful units that extract some operation and can be called multiple times with different inputs.

✓ What anonymous functions are

Anonymous functions are written in one line and can be called from anywhere within the same script. They are great for short operations like simple math.

6 – Summary

✓ L.O.1
✓ L.O.2
✓ L.O.3
✓ L.O.4
✓ L.O.5

✓ How to write anonymous functions

Anonymous functions are stored in a variable name and can accept input arguments, specified with `@(<inputs>)`, followed by the function declaration itself.

✓ How to determine when to use an anonymous function

Short, repeated operations are perfect for extracting into anonymous functions to make a more modular, and therefore more maintainable program.

6 – Summary

✓ L.O.1
✓ L.O.2
✓ L.O.3
✓ L.O.4
✓ L.O.5

✓ How to apply an anonymous function to the dot product

The dot product makes a great example because the math is simple, and it is also a very common operation in engineering. The inputs are both `1x3` vectors, covered in Lecture 1.4. The function then calculates and returns the scalar output.